CS-224 LAB 4
Preliminary Work

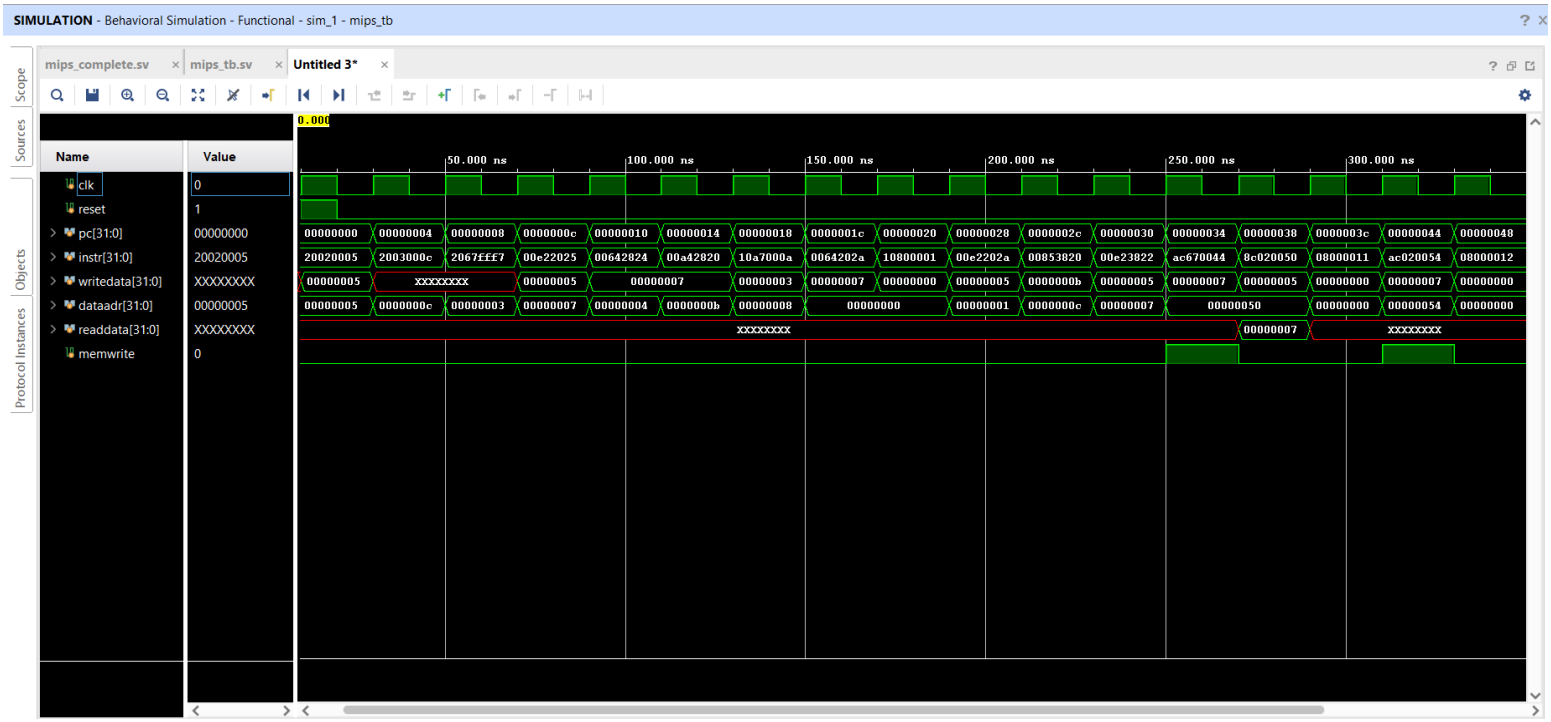
Full Name: Arda İynem
ID: 22002717
Section: 1
Lab No: 4
Date: 16/11/2022

Part 1

(a)

Location	Machine Instruction	Assembly Language
0x00000000	0x20020005	ADDI $v0 $zero 0x5
0x00000004	0x2003000C	ADDI $v1 $zero 0xC
0x00000008	0x2067FFF7	ADDI $a3 $v1 0xFFF7
0x0000000c	0x00E22025	OR $a0 $a3 $v0
0x00000010	0x00642824	AND $a1 $v1 $a0
0x00000014	0x00A42820	ADD $a1 $a1 $a0
0x00000018	0x10A7000A	BEQ $a1 $a3 0xA
0x0000001c	0x0064202A	SLT $a0 $v1 $a0
0x00000020	0x10800001	BEQ $a0 $zero 0x1
0x00000024	0x20050000	ADDI $a1 $zero 0x0
0x00000028	0x00E2202A	SLT $a0 $a3 $v0
0x0000002c	0x00853820	ADD $a3 $a0 $a1
0x00000030	0x00E23822	SUB $a3 $a3 $v0
0x00000034	0xAC670044	SW $a3 0x44($v1)
0x00000038	0x8C020050	LW $v0 0x50($zero)
0x0000003c	0x08000011	J 0x11
0x00000040	0x20020001	ADDI $v0 $zero 0x1
0x00000044	0xAC020054	SW $v0 0x54($zero)
0x00000048	0x08000012	J 0x12

(d)



(e)

i) In R type instructions *writedata* corresponds to the data at the register specified by rt.

ii) Because *writedata* corresponds to data at the register specified by rt ([20:16] bits of instruction is rt) and in these early I-type instructions rt is used as a destination instead of a source register, therefore registers has the value “XXXXXXXX” since they are not assigned a value before.

iii) *readdata* shows the value held in data memory at the address pointed by *dataadr* but since the ALU’s result (*dataadr*) isn’t a valid address at data memory most of the time -except the times that ALU is used specifically for memory operations like lw and sw- the result is “XXXXXXXX”. And even if the address is valid by chance, the data contained at that address might be uninitialized which will also return “XXXXXXXX”.

iv) In R type instructions *dataadr* corresponds to ALU’s result, and ALU’s sources are rs and rt as *ALUSrc* mux select = 0 with R-type instructions. Consequently, *dataadr* depends on the ALU operation chosen by *funct* bits.

v) In sw instructions *memwrite* becomes 1 as *writedata* value needs to be written on the memory address specified by *dataadr*.

(f) Modified ALU for a << b operation

```
module alu(input  logic [31:0] a, b,
           input  logic [2:0]  alucont,
           output logic [31:0] result,
           output logic zero);

    always_comb
        case(alucont)
            3'b010: result = a + b;
            3'b110: result = a - b;
            3'b000: result = a & b;
            3'b001: result = a | b;
            3'b111: result = (a < b) ? 1 : 0;
            3'b011: result = a << b[4:0]; // NEW
            default: result = {32{1'bx}};
        endcase

    assign zero = (result == 0) ? 1'b1 : 1'b0;
endmodule
```

Note: In this implementation, only the first 5 bits of b are taken for the shift amount since any shift amount more than 31 will result in 0 no matter what the value of a is which is unnecessary.

Part 2

(a)

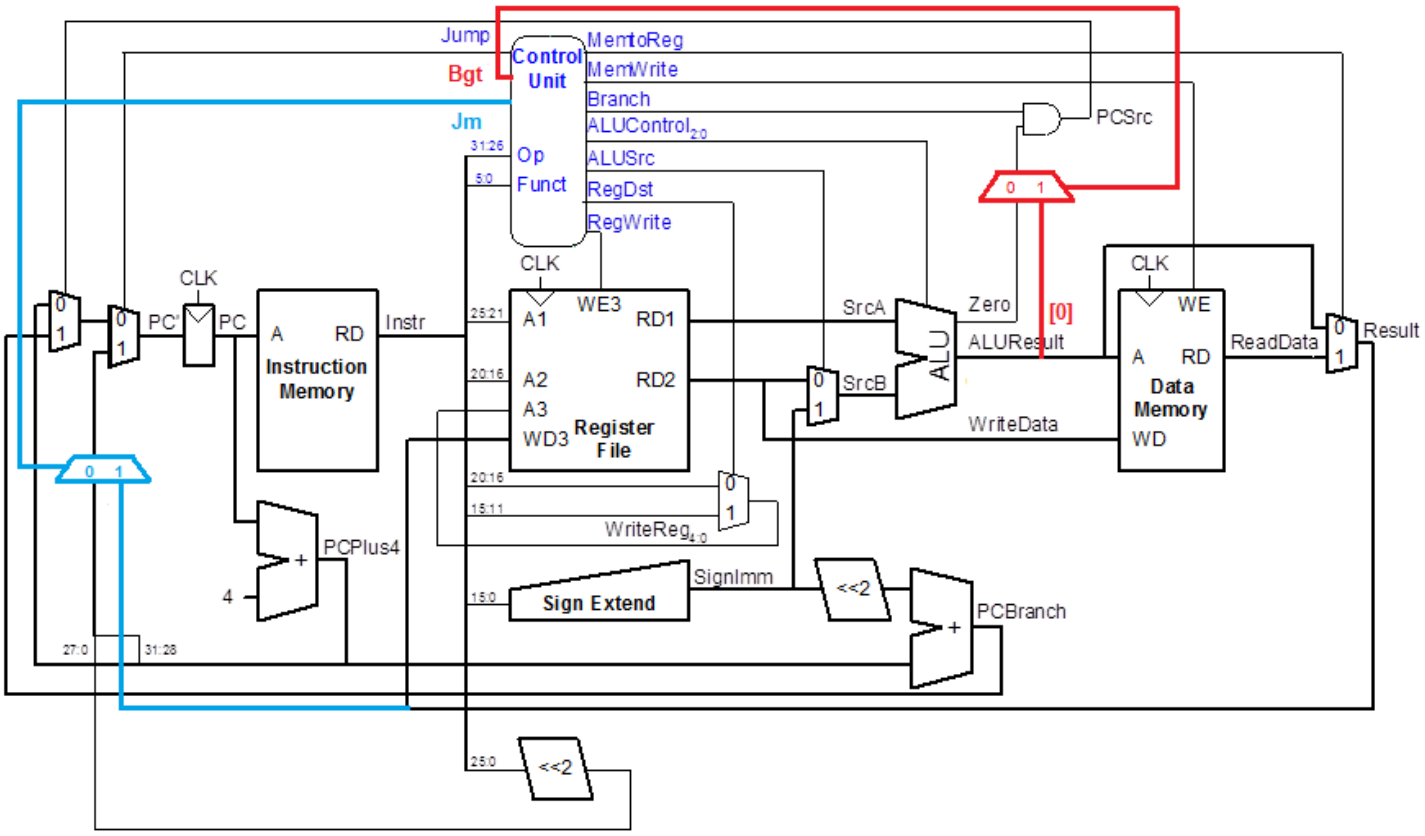
RTL Expression for **bgt**

IM[PC]
if (RF[rs] > RF[rt])
PC ← PC + 4 + {14{immediate[15]}, immediate, 2'b0 }

RTL Expression for **jm**

IM[PC]
PC ← DM[RF[rs] + {16{immediate[15]}, immediate}]

(b)



(c)

Modified Main Decoder

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump	Bgt	Jm
R-type	000000	1	1	0	0	0	0	10	0	X	X
lw	100011	1	0	1	0	0	1	00	0	X	X
sw	101011	0	X	1	0	1	X	00	0	X	X
beq	000100	0	X	0	1	0	X	01	0	0	X
addi	001000	1	0	1	0	0	0	00	0	X	X
j	000010	0	X	X	X	0	X	XX	1	X	0
bgt	000011	0	X	0	1	0	X	11	0	1	X
jm	000001	0	X	1	X	0	1	00	1	X	1

Modified ALU Decoder

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
10	100000 (add)	010 (add)
10	100010 (sub)	110 (subtract)
10	100100 (and)	000 (and)
10	100101 (or)	001 (or)
10	101010 (slt)	111 (set less than)
11	X	111 (set less than)